

Counting Reads

Dr. Princess Rodriguez

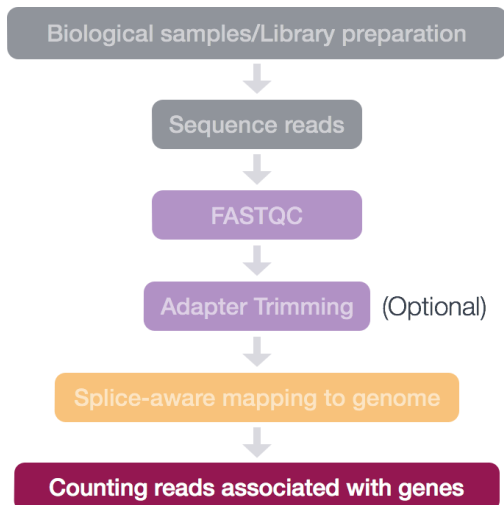
2026-03-02

Learning Objectives

- Understand the purpose of read counting in RNA-Seq analysis
- Describe how HTseq assigns reads to genomic features
- Distinguish between union, intersection-strict, and intersection-nonempty counting modes
- Interpret HTSeq output files

Counting reads as a measure of gene expression

Once we have our reads aligned to the genome, the next step is to count how many reads have mapped to each gene.



Gene Level vs Transcript Level Quantification

- In RNA-seq analysis, gene-level quantification summarizes expression by aggregating counts across all transcripts of a gene, while transcript-level quantification considers each transcript (isoform) separately, allowing for the detection of changes in isoform usage

Gene Level vs Transcript Level Quantification

- HTSeq report raw counts; counts are aggregated across all transcripts belonging to a single gene.
- Other tools (RSEM or SALMON) account for multiple transcripts per gene, assigning fractional counts instead of whole numbers. For example, if one read aligns to two transcripts, it may be counted as 0.5 for each transcript rather than a whole number.

Output of counting = A count matrix, with genes as rows and samples are columns

These are the “raw” counts and will be used in statistical programs downstream for differential gene expression.

Each column is a sample

Each row is a gene

GENE ID	KD.2	KD.3	OE.1	OE.2	OE.3	IR.1	IR.2	IR.3
1/2-SBSRNA4	57	41	64	55	38	45	31	39
A1BG	71	40	100	81	41	77	58	40
A1BG-AS1	256	177	220	189	107	213	172	126
A1CF	0	1	1	0	0	0	0	0
A2LD1	146	81	138	125	52	91	80	50
A2M	10	9	2	5	2	9	8	4
A2ML1	3	2	6	5	2	2	1	0
A2MP1	0	0	2	1	3	0	2	1
A4GALT	56	37	107	118	65	49	52	37
A4GNT	0	0	0	0	1	0	0	0
AA06	0	0	0	0	0	0	0	0
AAA1	0	0	1	0	0	0	0	0
AAAS	2288	1363	1753	1727	835	1672	1389	1121
AACS	1586	923	951	967	484	938	771	635
AACSP1	1	1	3	0	1	1	1	3
AADAC	0	0	0	0	0	0	0	0
AADACL2	0	0	0	0	0	0	0	0
AADACL3	0	0	0	0	0	0	0	0
AADACL4	0	0	1	1	0	0	0	0
AADAT	856	539	593	576	359	567	521	416
AAGAB	4648	2550	2648	2356	1481	3265	2790	2118

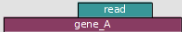
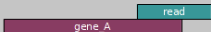

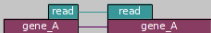

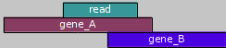
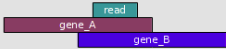

HTSeq-count

HTSeq-count is a command-line tool used in RNA-Seq data analysis to count the number of sequencing reads that align to specific genomic features, such as genes. It is part of the HTSeq Python package and is commonly used in differential gene expression analysis pipelines.

What is a feature?

- A common task is to count how many reads map to each feature.
- A feature refers to a specific interval (i.e., a range of positions) on a chromosome or a union of such intervals.
- Since our example data comes from an RNA-Seq experiment, we aim to count how many reads fall within the exonic regions of each gene.

How does htseq-count manage reads that align to multiple features?

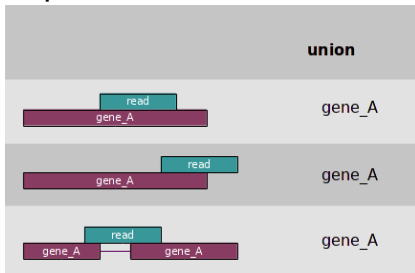
	union	intersection_strict	intersection_nonempty
	gene_A	gene_A	gene_A
	gene_A	no_feature	gene_A
	gene_A	no_feature	gene_A
	gene_A	gene_A	gene_A
	gene_A	gene_A	gene_A
	ambiguous (both genes with --nonunique all)	gene_A	gene_A
	ambiguous (both genes with --nonunique all)		
	alignment_not_unique		

How does htseq-count manage reads that align to multiple features?

The htseq-count script offers **three different modes** to handle such cases:

● Union (Recommended for most cases):

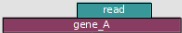
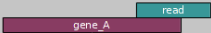

- A read is assigned to a feature if any part of the read overlaps with it.
- If a read overlaps multiple features, it is not counted at all (to avoid ambiguity).



How does htseq-count manage reads that align to multiple features?

Intersection-strict:

- A read is assigned to a feature only if every position of the read overlaps with that feature.
- If a read overlaps multiple features but not completely within one, it is not counted.

	union	intersection_strict
	gene_A	gene_A
	gene_A	no_feature
	gene_A	no_feature

How does `htseq-count` manage reads that align to multiple features?

Intersection-nonempty:

- A read is assigned to a feature only if it overlaps with at least one feature at every position.
- Unlike `intersection-strict`, it ignores positions where no features are present.

Union mode is the most commonly used in HTSeq

	union	intersection_strict	intersection_nonempty
	gene_A	gene_A	gene_A
	gene_A	no_feature	gene_A
	gene_A	no_feature	gene_A
	gene_A	gene_A	gene_A
	gene_A	gene_A	gene_A
	ambiguous (both genes with --nonunique all)	gene_A	gene_A
	ambiguous (both genes with --nonunique all)		

Running htseq-count

Begin by loading the htseq-count module with `module load`

```
module load gcc/13.3.0-xp3epyt  
py-htseq/2.0.3-mb7ap7s
```

Run the following to check the module is loaded:

```
htseq-count --help
```

Basic Command Syntax

```
htseq-count -f bam -s no -i gene_id sample1.bam genes.gtf
```

- f bam → Specifies input format (can be sam or bam).
- s no → Defines strandedness (yes, no, or reverse).
- i gene_id → used to identify the GTF feature attribute (gene_name or gene_id)
- aligned_reads.bam → Input file with mapped reads.
- genes.gtf → Reference annotation file.
- > gene_counts.txt → Redirects output to a file.

What happens if strandedness is set incorrectly?

If wrong, HTSeq will either assign reads to the wrong strand or most reads will be ignored

How can you diagnose if this is the case?

- Will have an extremely high `__no__feature`
- Very low gene counts

GFF/GTF attribute

`-i <id attribute>`, `--idattr=<id attribute>`

The feature ID is used to identify the counts in the output table. The default, suitable for RNA-Seq analysis using an Ensembl GTF file, is `gene_id`.

Limitations & Alternatives

- HTSeq-count is a gene-level summarization tool; it does not handle transcript-level quantification.
- It requires sorted BAM/SAM files.
- Alternative tools like featureCounts (...which is faster) or Salmon/RSEM (for transcript-level quantification) may be preferred in some cases.

Review Class Exercise Folder Content

Class Exercise: Running RSeQC”

Copy Folder: Make a copy of the following folder into your home directory:

```
/gpfs1/cl/mmg3320/course_materials/htseq_demo
```

2 **Determine Strandedness:** You will need to run RSeQC to determine strandedness of the demo data. There are three options to select from:

- `-s yes`, reads are mapped to the same strand as the sense strand
- `-s no`, reads can map to either strand (unstranded)
- `-s reverse`, reads are mapped to the opposite strand (anti-sense)

3 **Interpret the multiqc output**

Let's look at the
multiqc output

Class Exercise 2: Running HTSeq-count

Modify `htseq-count.sh` script:

- 2** **Submit the `htseq-count.sh` script** after modifying it. This should only take a few minutes.
- 3** **Look inside** of the `htseq-count_XXXXXX.out` file after the job is completed. It should appear identical as below:

Class Exercise 3: MultiQC

Read the section below titled `htseq-count` output

- 2 Generate a final multiqc output; this time do so inside of the `bams\` folder.

Let's look at the
multiqc output

htseq-count output

The output of htseq-count consists of two main files for each sample:

- A summary/log file

- A counts file that lists the number of reads mapped to each gene or feature. This is a tab-delimited file with gene IDs and their associated read counts.

head KO_hg19_rep2_sorted.gene_id.count.txt

```
AADACL3 0
AADACL4 0
ABCA4 0
ABCB10 0
ABCD3 1
```

htseq-count output (tail)

```
tail KO_hg19_rep2_sorted.gene_id.count.txt
```

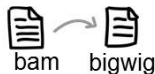
```
__no_feature      32813  
__ambiguous       4372  
__too_low_aQual   0  
__not_aligned    15209  
__alignment_not_unique 3667
```

What to “aim” for in HTSeq Count

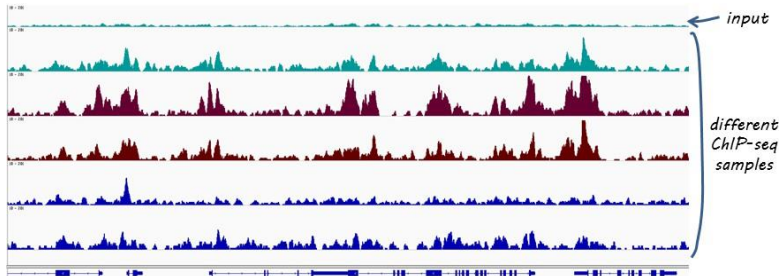
1. Total reads Processed ~25 million
2. Percentage Assigned to Genes (70-90%)
3. No_feature is ideally less than ~25%

Creating bigWig files

- We will now take our BAM files and convert them into bigWig files. The bigWig format is an indexed binary format useful for dense, continuous data that can be displayed in a genome browser as a graph/track.
- To create bigWig files we will use [deepTools](#), a suite of Python tools developed for the efficient analysis of high-throughput sequencing data, such as CHIP-seq, RNA-seq or MNase-seq. `deepTools` has a wide variety of commands that go beyond what we will cover today.



for visualizing continuous data, e.g. in the UCSC Genome Browser or IGV, bigWig files come in really handy



remember that there are 2 deepTools for bam \rightarrow bigWig conversion:

- ❖ **bamCoverage**: for individual files (like those shown here)
- ❖ **bamCompare**: to normalize two files to each other

Figure 5: BigWig Tracks

Checking/Creating index file for the BAM file

- Often, when working with BAM files you will find that many tools require an index (an associated `.bai` file). Indexing the BAM file aims to achieve fast retrieval of alignments overlapping a specified region without going through the whole alignment file. Essentially, a `bai` file along with the `bam` ensures that downstream applications are able to use the information with the `bam` file much more speedily.
- As a reminder, we used [SAMtools](#), specifically the `samtools index` command, to index the BAM files.

bamCoverage from deepTools

This command takes a **BAM file as input** and evaluates which areas of the genome have reads associated with them, i.e. how much of the genome is “covered” with reads. The coverage is calculated as the number of reads per bin, where bins are short consecutive sections of the genome (bins) that can be defined by the user. The **output of this command is a bigWig file.**

These are some parameters of `bamCoverage` that are worth considering:

- `normalizeUsing`: Possible choices: RPKM, CPM, BPM, RPGC. By default, no normalization is applied. More on this below.
- `binSize`: size of bins in bases (default is 50)
- `--effectiveGenomeSize`: the portion of the genome that is mappable. It is useful to consider this when computing your scaling factor..

DeepTools offers different **methods of normalization**

- Reads Per Kilobase per Million mapped reads (RPKM)
 - number of reads per bin / (number of mapped reads (in millions) * bin length (kb))
- Counts per million (CPM); this is similar to CPM in RNA-seq
 - number of reads per bin / number of mapped reads (in millions)
- Bins Per Million mapped reads (BPM); same as TPM in RNA-seq
 - number of reads per bin / sum of all reads per bin (in millions)

Class Exercise: Running deeptools/bamCoverage

We will be using the bare minimum of parameters as shown in the code below.

Create a bigWig file for KO_hg19_rep2_sorted.bam and WT_hg19_rep2_sorted.bam:

```
module load deeptools/3.5.5
```

```
bamCoverage -b KO_hg19_rep2.bam -o KO_hg19_rep2.bw
```

Note: Normally, this command can take up to 10 minutes to complete.

Visualize with IGV:

- Start IGV, You may have this previously installed on your laptop. If not no worries, use the [IGV Web App](#).
- Load the Human genome (hg19) into IGV using the dropdown menu at the top left of your screen.
- Load the .bw file using the “Load from File...” or “Tracks” option.
- Type MOV10 into the search bar.

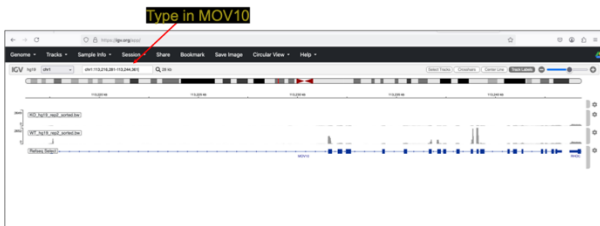


Figure 6: shells

Citation

This lesson has been developed by members of the teaching team at the [Harvard Chan Bioinformatics Core \(HBC\)](#). These are open access materials distributed under the terms of the [Creative Commons Attribution license](#) (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.